

Master Coding Interview Questions (+DSA)

P.S -You can convert the code to Python also by simple prompt in ChatGPT

<https://chat.openai.com/share/884ea526-e131-4b0b-a63e-d2ba87d95f11>

Prompt - Convert the below program to Python 3.11 version "Paste Code"

✓ How to Take Input from Users?

- You can take input from users using the Scanner class

```
import java.util.Scanner;

public class UserInputExample {
    public static void main(String[] args) {
        // Create a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user for input
        System.out.print("Enter your name: ");
        // Read a line of text entered by the user
        String name = scanner.nextLine();
        // Prompt the user for a number
        System.out.print("Enter your age: ");
        // Read an integer entered by the user
        int age = scanner.nextInt();
        // Display the user's input
        System.out.println("Hello, " + name + "! You are " + age + "
years old.");
        // Close the Scanner to release resources (optional)
        scanner.close();
    }
}
```

✓ Table Print using System.out.printf

Print lines of output; each line (where) contains the of in the form:

$N \times i = \text{result}$.

Sample Input

2

Sample Output

-
- 2 x 1 = 2
- 2 x 2 = 4
- 2 x 3 = 6
- 2 x 4 = 8
- 2 x 5 = 10
- 2 x 6 = 12
- 2 x 7 = 14
- 2 x 8 = 16
- 2 x 9 = 18
- 2 x 10 = 20

```
int n = 5;
for (int i = 1; i <= 10; i++) {
    System.out.printf("%d\t%d\t%d\n", i+"x"+n+"=", n*i);
}
```

✅ FizzBuzz Test:

Write a program that prints numbers from 1 to 100. However, for multiples of 3, print "Fizz" instead of the number, and for multiples of 5, print "Buzz." For numbers that are multiples of both 3 and 5, print "FizzBuzz."

```
public class FizzBuzz {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 100; i++) {  
            if (i % 3 == 0 && i % 5 == 0) {  
                System.out.println("FizzBuzz");  
            } else if (i % 3 == 0) {  
                System.out.println("Fizz");  
            } else if (i % 5 == 0) {  
                System.out.println("Buzz");  
            } else {  
                System.out.println(i);  
            }  
        }  
    }  
}
```

✅ Palindrome Checker:

Create a program that checks whether a given string is a palindrome. A palindrome is a word or phrase that reads the same backward as forward (ignoring spaces, punctuation, and capitalization). Use an if-else statement to determine if the string is a palindrome.

✓ Grade Calculator:

Write a program that calculates and displays the letter grade for a given numerical score (e.g., A, B, C, D, or F) based on the following grading scale:

A: 90-100

B: 80-89

C: 70-79

D: 60-69

F: 0-59

```
public class GradeCalculator {
    public static void main(String[] args) {
        int score = 85; // Replace with your numerical score
        char grade;

        if (score >= 90 && score <= 100) {
            grade = 'A';
        } else if (score >= 80 && score < 90) {
            grade = 'B';
        } else if (score >= 70 && score < 80) {
            grade = 'C';
        } else if (score >= 60 && score < 70) {
            grade = 'D';
        } else {
            grade = 'F';
        }

        System.out.println("Your grade is: " + grade);
    }
}
```

✓ Leap Year Checker:

Create a program that determines whether a given year is a leap year. A leap year is divisible by 4, but not by 100 unless it is also divisible by 400. Use an if-else statement to make this determination.

```
public class LeapYearChecker {
    public static void main(String[] args) {
        int year = 2024; // Replace with the year you want to check
    }
}
```

```

        boolean isLeapYear = false;

        if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
            isLeapYear = true;
        }

        if (isLeapYear) {
            System.out.println(year + " is a leap year.");
        } else {
            System.out.println(year + " is not a leap year.");
        }
    }
}

```

✓ Triangle Classifier:

Write a program that classifies a triangle based on its side lengths. Given three input values representing the lengths of the sides, determine if the triangle is equilateral (all sides are equal), isosceles (exactly two sides are equal), or scalene (no sides are equal). Use an if-else statement to classify the triangle.

```

public class TriangleClassifier {
    public static void main(String[] args) {
        int side1 = 5; // Replace with the lengths of your triangle's
        sides
        int side2 = 4;
        int side3 = 4;

        if (side1 == side2 && side2 == side3) {
            System.out.println("Equilateral triangle");
        } else if (side1 == side2 || side1 == side3 || side2 == side3) {
            System.out.println("Isosceles triangle");
        } else {
            System.out.println("Scalene triangle");
        }
    }
}

```

✓ Right Triangle Star Pattern

```
*  
**  
***  
****  
*****
```

```
int n = 5;  
for (int i = 1; i <= n; i++) {  
    for (int j = 1; j <= i; j++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

✓ Left Triangle Star Pattern

```
*****  
****  
***  
**  
*
```

```
int n = 5;  
for (int i = n; i >= 1; i--) {  
    for (int j = 1; j <= i; j++) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

✓ Pyramid pattern in Java

```
*  
**  
***  
****  
*****  
*****  
*****
```

```

public class PyramidPattern {
    public static void main(String[] args) {
        int rows = 5; // Number of rows in the pyramid

        for (int i = 1; i <= rows; i++) {
            // Print spaces before the stars
            for (int j = 1; j <= rows - i; j++) {
                System.out.print(" ");
            }

            // Print stars
            for (int k = 1; k <= 2 * i - 1; k++) {
                System.out.print("*");
            }

            // Move to the next line
            System.out.println();
        }
    }
}

```

✓ Count vowels and consonants in a String

```

public int[] countVowelsCons(String str) {
    int vCount = 0;
    int cCount = 0;

    for(int i=0; i<str.length(); i++) {
        char ch = str.charAt(i);
        if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            vCount++;
        } else if((ch >= 'a' && ch <= 'z')) {
            cCount++;
        }
    }

    return new int[]{vCount, cCount};
}

```

✓ Prime Nuber checker

some valid and invalid test cases for the prime number checker in Java:

Valid Test Cases (Prime Numbers):



1. `num = 2` - This is the smallest prime number.
2. `num = 7` - A prime number greater than 2.
3. `num = 13` - Another prime number.
4. `num = 97` - A larger prime number.
5. `num = 101` - Another example of a prime number.

Invalid Test Cases (Non-Prime Numbers):

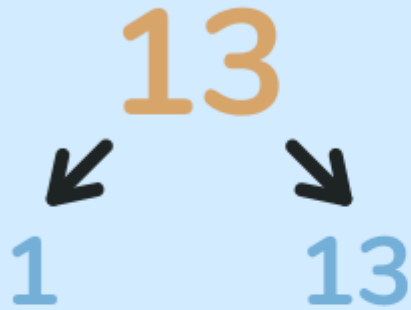
1. `num = 1` - 1 is not considered a prime number.
2. `num = 4` - It's divisible by 2, so it's not prime.
3. `num = 8` - Also divisible by 2.
4. `num = 15` - Divisible by 3 and 5.
5. `num = 100` - Divisible by 2 and 5.

Definition

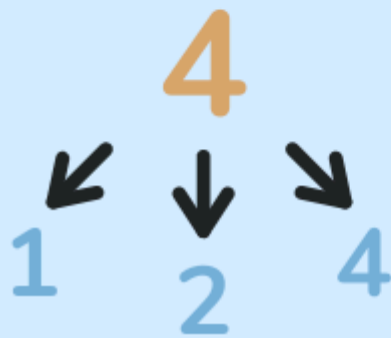
Prime Number	A positive integer with only two factors, 1 and itself. A number that is not composite.
Examples of Prime Numbers: 3 has factors 1, 3 5 has factors 1, 5 13 has factors 1, 13 23 has factors 1, 23	Examples of Composite Numbers: 4 has factors 1, 2, 4 8 has factors 1, 2, 4, 8 24 has factors 1, 2, 3, 4, 6, 8, 12, 24 15 has factors 1, 3, 5, 15



How do prime numbers work?



13 has **only two factors** - itself and 1. So it is a prime number.



4 has **three factors** - itself, 1 and 2. So it is NOT a prime number.

```
import java.util.Scanner;

public class PrimeChecker {
    public static boolean isPrime(int number) {
        if (number <= 1) {
            return false;
        }
        if (number <= 3) {
            return true;
        }
        if (number % 2 == 0 || number % 3 == 0) {
            return false;
        }
        for (int i = 5; i * i <= number; i += 6) {
            if (number % i == 0 || number % (i + 2) == 0) {
                return false;
            }
        }
    }
}
```

```

        return true;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        scanner.close();

        if (isPrime(num)) {
            System.out.println(num + " is a prime number.");
        } else {
            System.out.println(num + " is not a prime number.");
        }
    }
}

```

✓ Palindrome of String

```

public class PalindromeCheck {

    public static boolean isPalindrome(String str) {

        str = str.toLowerCase();

        for(int i=0; i<str.length()/2; i++) {
            char start = str.charAt(i);
            char end = str.charAt(str.length()-1-i);

            if(start != end) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        String input = "madam";
        if(isPalindrome(input)) {
            System.out.println("Palindrome");
        }
    }
}

```

```

    } else {
        System.out.println("Not Palindrome");
    }
}
}

```

String Reverse

Given a string , Reverse it and print true if they are same.
Madam is palindrome, pramod is not

Here is the full program to reverse words in-place in a given sentence using user input:

```
``java
```

```

import java.util.Scanner;

public class Main {

    public static void reverseWords(char[] sentence) {
        // Reverse entire sentence
        reverse(sentence, 0, sentence.length-1);

        int start = 0;
        for(int end=0; end<sentence.length; end++) {
            if(sentence[end] == ' ') {
                // Reverse current word
                reverse(sentence, start, end-1);
                start = end + 1;
            }
        }

        // Reverse last word
        reverse(sentence, start, sentence.length-1);
    }

    public static void reverse(char[] arr, int i, int j) {
        while(i < j) {
            char temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}

```

```

        arr[j] = temp;
        i++;
        j--;
    }
}

```

```

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a sentence: ");
    String input = scanner.nextLine();

    char[] sentence = input.toCharArray();
    reverseWords(sentence);

    System.out.println(new String(sentence));
}
}

```

Explanation:

- Convert sentence to char array
- Reverse entire sentence first
- Then reverse each word by detecting word boundaries
- Reverse last word separately after loop
- Print reversed char array

This reverses the words in-place without using extra space.

Let me know if you have any other questions!

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");
        String input = scanner.nextLine();
    }
}

```

```

String reversed = new StringBuilder(input).reverse().toString();

if(input.equals(reversed)) {
    System.out.println("Yes");
} else {
    System.out.println("No");
}
}
}

```

✓ Duplicate String

```

public class RemoveDuplicates {

    public static String removeDuplicates(String str) {

        String result = "";

        for(int i=0; i<str.length(); i++) {
            char ch = str.charAt(i);
            if(result.indexOf(ch) == -1) {
                result += ch;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        String input = "Hello World";
        String output = removeDuplicates(input);
        System.out.println(output);
    }
}

```

Advance using LinkedHashSet

```

public class RemoveDuplicates {

    public static String remove(String str) {
        // add chars to LinkedHashSet
    }

    public static void main(String[] args) {
        String input = "Hello World";
        String output = remove(input);
        System.out.println(output); // Helo Wrld
    }

}

```

Java Anagrams

```

String s1 = "silent";
String s2 = "listen";

if (areAnagrams(s1, s2)) {
    System.out.println("Anagrams");
} else {
    System.out.println("Not Anagrams");
}

```

```

public static boolean areAnagrams(String s1, String s2) {

    // Remove non-letter characters and convert to lowercase
    s1 = s1.replaceAll("[^a-zA-Z]", "").toLowerCase();
    s2 = s2.replaceAll("[^a-zA-Z]", "").toLowerCase();

    // Check if lengths are different
    if (s1.length() != s2.length()) {
        return false;
    }

    // Sort the characters of the strings
    char[] arr1 = s1.toCharArray();
    Arrays.sort(arr1);
    char[] arr2 = s2.toCharArray();
    Arrays.sort(arr2);

```

```

// Compare sorted strings
for (int i = 0; i < arr1.length; i++) {
    if (arr1[i] != arr2[i]) {
        return false;
    }
}

return true;
}

```

Another approach

```

public static boolean areAnagrams(String s1, String s2){

    // Remove non-letters and convert to same case
    s1 = s1.replaceAll("[^a-zA-Z]", "").toLowerCase();
    s2 = s2.replaceAll("[^a-zA-Z]", "").toLowerCase();

    // Check if lengths are different
    if(s1.length() != s2.length()){
        return false;
    }

    // Count frequency of characters
    int[] charFreq = new int[26];
    for(int i=0; i<s1.length(); i++){
        charFreq[s1.charAt(i) - 'a']++;
        charFreq[s2.charAt(i) - 'a']--;
    }

    // Check if all counts are 0
    for(int count : charFreq){
        if(count != 0){
            return false;
        }
    }
    return true;
}

```

✓ Valid Email Regex Java

pramod@live.com - Valid

pramod@dasda - Not valid

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class EmailValidation {

    public static void main(String[] args) {

        String email = "john@example.com";

        String emailRegex = "^[a-zA-Z0-9_+&*-]+(?:\\.|"+
            "[a-zA-Z0-9_+&*-]+)*@" +
            "(?:[a-zA-Z0-9-]+\\.|"+
            "[A-Z]{2,7})$";

        Pattern pat = Pattern.compile(emailRegex);
        Matcher mat = pat.matcher(email);

        if (mat.matches())
            System.out.println("Valid email");
        else
            System.out.println("Invalid email");

    }
}
```

Test for Invalid Emails

- "plainaddress" - Missing @ and domain
- "@invalid.com" - Missing username
- "joe@[123.123.123.123]" - Square brackets around IP address
- "näme@example.com" - Contains illegal character
- "john..doe@example.com" - Double dot
- "john@doe@example.com" - Double @
- "john@example@com" - Missing top level domain (.com)
- "john@example.c" - Top level domain too short
- "john@example.com1" - Top level domain cannot have numbers
- "<john@example..com>" - Double dot in domain name
- "john@example.com." - Trailing dot in domain name
- "<john@example..com>." - Double dots and trailing dot
- "john@example.com.." - Double dot before top level domain

✓ Exceptions

1. What is an Exception in Java?

Answer:

An exception is an event that disrupts the normal flow of a program. It is an object which is thrown at runtime. Exceptions are categorized into two types:

- Checked exceptions: These are checked at compile time. For example, IOException, SQLException.
- Unchecked exceptions: These are not checked at compile time. For example, NullPointerException, ArithmeticException.

2. What is the difference between checked and unchecked exceptions?

Answer:

- Checked exceptions: These must be either caught or declared in the method signature using 'throws'. They are checked by the compiler at compile time. Example: IOException.
- Unchecked exceptions: These do not need to be declared or caught. They are checked at runtime. Example: NullPointerException.

3. How do you handle exceptions in Java?

Answer:

Exceptions in Java are handled using the try-catch-finally blocks:

```
try {  
    // code that may throw an exception  
} catch (ExceptionType1 e1) {  
    // handle exception e1  
} catch (ExceptionType2 e2) {  
    // handle exception e2  
} finally {  
    // code that will always execute  
}
```

The finally block is optional and is used for cleanup code, which is executed irrespective of whether an exception is thrown or not.

4. What is the purpose of the finally block?

Answer:

The finally block is used to execute important code such as closing resources, regardless of whether an exception occurs or not. It always executes when the try block exits, except if the JVM exits abruptly.

5. What is a throw keyword in Java?

Answer:

The throw keyword is used to explicitly throw an exception. For example:

```
if (num < 0) {  
    throw new IllegalArgumentException("Number must be non-negative");  
}
```

```
}
```

6. What is the throws keyword in Java?

Answer:

The throws keyword is used in the method signature to declare that a method might throw one or more exceptions. It informs the caller of the method to handle these exceptions. For example:

```
public void readFile(String fileName) throws IOException {  
    // code that might throw IOException  
}
```

7. What is a custom exception and how do you create one in Java?

Answer:

A custom exception is a user-defined exception that extends Exception or RuntimeException. For example:

```
public class MyCustomException extends Exception {  
    public MyCustomException(String message) {  
        super(message);  
    }  
}
```

To throw this exception:

```
if (condition) {  
    throw new MyCustomException("Custom exception occurred");  
}
```

8. Explain the concept of exception chaining in Java.

Answer:

Exception chaining is a technique where one exception is used to cause another. It is useful for maintaining the stack trace of the original exception. This is done using constructors in Throwable class that accept a cause.

```
try {  
    // code that causes an exception  
} catch (Exception e) {  
    throw new MyCustomException("Custom exception occurred", e);  
}
```

9. What are common Selenium exceptions and how do you handle them?

Answer:

Some common Selenium exceptions include:

- NoSuchElementException: Element not found in the DOM.
- TimeoutException: Command did not complete in the specified time.
- WebDriverException: Generic WebDriver error.

Handling these exceptions involves using try-catch blocks and ensuring appropriate wait strategies (like explicit waits) to avoid these issues.

```
try {  
    WebElement element = driver.findElement(By.id("nonexistent-id"));  
} catch (NoSuchElementException e) {
```

```
    System.out.println("Element not found: " + e.getMessage());
}
```

10. How do you use try-with-resources statement in Java?

Answer:

The try-with-resources statement is used to automatically close resources (like files, sockets) that implement the AutoCloseable interface. For example:

```
try (BufferedReader br = new BufferedReader(new FileReader("file.txt")))
{
    // read file
} catch (IOException e) {
    e.printStackTrace();
}
```

The BufferedReader will be closed automatically at the end of the try block.

✓ File Read and Token it

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;

public class FileReadAndTokenize {
    public static void main(String[] args) {
        String filePath = "path/to/your/file.txt"; // Change to your
        // file path
        BufferedReader reader = null;

        try {
            reader = new BufferedReader(new FileReader(filePath));
            String line;
            while ((line = reader.readLine()) != null) {
                tokenizeLine(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (reader != null) reader.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

```

private static void tokenizeLine(String line) {
    StringTokenizer tokenizer = new StringTokenizer(line, " ,.?!?");

    while (tokenizer.hasMoreTokens()) {
        String token = tokenizer.nextToken();
        System.out.println(token);
    }
}
}

```

✓ List Questions

Explain the difference between List, Set, and Map in Java.

List: An ordered collection that allows duplicate elements. It maintains the insertion order.
Examples: ArrayList, LinkedList

Set: A collection that does not allow duplicate elements. It does not maintain any order (except LinkedHashSet which maintains insertion order and TreeSet which maintains a sorted order).

Examples: HashSet, LinkedHashSet, TreeSet

Map: A collection that maps keys to values, with no duplicate keys allowed. It does not maintain any order (except LinkedHashMap which maintains insertion order and TreeMap which maintains a sorted order).

Examples: HashMap, LinkedHashMap, TreeMap

How do you reverse an ArrayList in Java?

```

import java.util.*;

public class ReverseArrayList {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
        Collections.reverse(list);
        System.out.println("Reversed ArrayList: " + list);
    }
}

```

How do you merge two sorted lists into one sorted list?

```
import java.util.*;

public class MergeSortedLists {
    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 3, 5, 7));
        List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 4, 6, 8));
        List<Integer> mergedList = mergeSortedLists(list1, list2);
        System.out.println("Merged Sorted List: " + mergedList);
    }

    public static List<Integer> mergeSortedLists(List<Integer> list1, List<Integer> list2) {
        List<Integer> result = new ArrayList<>();
        int i = 0, j = 0;
        while (i < list1.size() && j < list2.size()) {
            if (list1.get(i) < list2.get(j)) {
                result.add(list1.get(i++));
            } else {
                result.add(list2.get(j++));
            }
        }
        while (i < list1.size()) {
            result.add(list1.get(i++));
        }
        while (j < list2.size()) {
            result.add(list2.get(j++));
        }
        return result;
    }
}
```

How do you find the most frequent element in a list?

```
import java.util.*;

public class MostFrequentElement {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 2, 3,
```

```

3, 3, 4, 4, 4, 4));
    int mostFrequent = findMostFrequentElement(list);
    System.out.println("Most Frequent Element: " + mostFrequent);
}

public static int findMostFrequentElement(List<Integer> list) {
    Map<Integer, Integer> frequencyMap = new HashMap<>();
    for (int num : list) {
        frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) +
1);
    }
    return Collections.max(frequencyMap.entrySet(),
Map.Entry.comparingByValue()).getKey();
}
}

```

How do you iterate through a LinkedList?

```

import java.util.*;

public class IterateLinkedList {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>(Arrays.asList("A",
"B", "C", "D"));

        // Using Iterator
        Iterator<String> iterator = list.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        // Using enhanced for loop
        for (String item : list) {
            System.out.println(item);
        }

        // Using for loop with index
        for (int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i));
        }
    }
}

```

Write a function to find the intersection of two lists.

```
import java.util.*;

public class ListIntersection {
    public static void main(String[] args) {
        List<Integer> list1 = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
        List<Integer> list2 = new ArrayList<>(Arrays.asList(4, 5, 6, 7, 8));
        List<Integer> intersection = findIntersection(list1, list2);
        System.out.println("Intersection: " + intersection);
    }

    public static List<Integer> findIntersection(List<Integer> list1, List<Integer> list2) {
        Set<Integer> set1 = new HashSet<>(list1);
        Set<Integer> set2 = new HashSet<>(list2);
        set1.retainAll(set2);
        return new ArrayList<>(set1);
    }
}
```

How do you sort a LinkedList of custom objects by a specific field?

```
import java.util.*;

public class SortLinkedList {
    public static void main(String[] args) {
        List<Person> list = new LinkedList<>(Arrays.asList(
            new Person("John", 25),
            new Person("Jane", 22),
            new Person("Tom", 28)
        ));
        Collections.sort(list, Comparator.comparingInt(Person::getAge));
        for (Person person : list) {
            System.out.println(person.getName() + " - " + person.getAge());
        }
    }
}
```

```
class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

How do you convert a List to a Set?

```
import java.util.*;
```

```
public class ListToSet {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>(Arrays.asList("apple",
"banana", "orange", "apple"));
        Set<String> set = new HashSet<>(list);

        System.out.println("List: " + list);
        System.out.println("Set: " + set);
    }
}
```

How do you remove duplicates from a List?

```
import java.util.*;

public class RemoveDuplicates {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>(Arrays.asList("apple",
"banana", "orange", "apple"));
    }
}
```



```

        Set<String> set = new HashSet<>(list);
        list.clear();
        list.addAll(set);

        System.out.println("List without duplicates: " + list);
    }
}

```

Write a program to sort a List of strings in Java.

```

import java.util.*;

public class SortList {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>(Arrays.asList("banana",
"apple", "orange"));
        Collections.sort(list);

        System.out.println("Sorted List: " + list);
    }
}

```

✓ Hash map

How do you iterate over a Map in Java?

```

import java.util.*;

public class IterateMap {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        map.put("apple", 1);
        map.put("banana", 2);
        map.put("orange", 3);

        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            System.out.println(entry.getKey() + ": " +
entry.getValue());
        }
    }
}

```

Two Sum Problem Statement

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

```
import java.util.HashMap;
import java.util.Map;

public class TwoSum {
    public static void main(String[] args) {
        int[] nums = {2, 7, 11, 15};
        int target = 9;
        int[] result = twoSum(nums, target);

        if (result != null) {
            System.out.println("Indices: [" + result[0] + ", " +
result[1] + "]");
        } else {
            System.out.println("No solution found.");
        }
    }

    public static int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];

            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }

            map.put(nums[i], i);
        }

        // In case there is no solution, we'll just return null
        return null;
    }
}
```

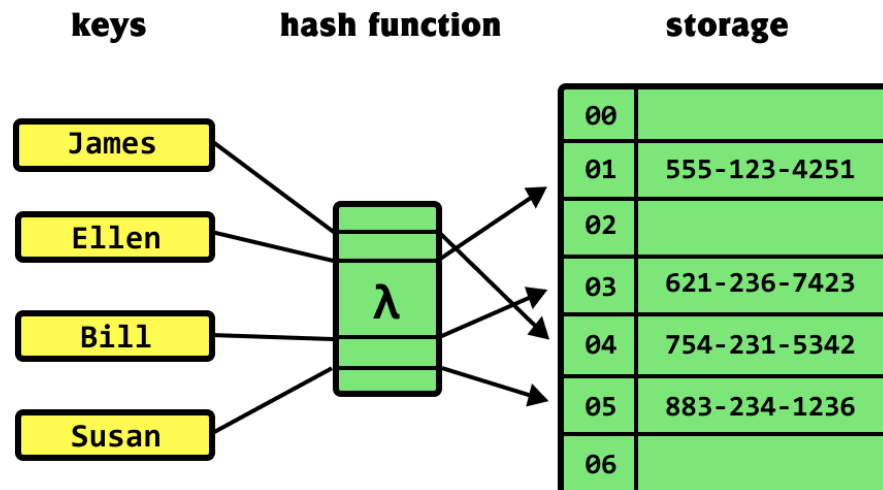
✓ Hashtable

What is a Hashtable in Java, and how does it differ from HashMap?

A hash table is a data structure that is used to store keys/value pairs. It uses a hash function to compute an index into an array in which an element will be inserted or searched. [Ref](#)

```
// Hashtable is synchronized, HashMap is not
Hashtable<Integer, String> hashtable = new Hashtable<>();
HashMap<Integer, String> hashMap = new HashMap<>();
```

Explain the internal working of a Hashtable.



Hashtable stores key-value pairs in an array of buckets. Each bucket contains a linked list of entries to handle collisions.

How would you iterate through the entries of a Hashtable?

```
Hashtable<Integer, String> hashtable = new Hashtable<>();
hashtable.put(1, "one");
hashtable.put(2, "two");

Enumeration<Integer> keys = hashtable.keys();
while (keys.hasMoreElements()) {
    Integer key = keys.nextElement();
```

```
        System.out.println("Key: " + key + " Value: " + hashtable.get(key));
    }

    // Alternatively, using entrySet()
    for (Map.Entry<Integer, String> entry : hashtable.entrySet()) {
        System.out.println("Key: " + entry.getKey() + " Value: " +
            entry.getValue());
    }
```

How does Hashtable ensure thread safety? Discuss its methods like put() and get()?

```
public synchronized V put(K key, V value) {
    // Synchronized block ensures thread safety
    ...
}

public synchronized V get(Object key) {
    // Synchronized block ensures thread safety
    ...
}
```

Hash Table

Here all strings are sorted at same index

Index				
0				
1				
2	abcdef	bcdefa	cdefab	defabc
3				
4				
-				
-				
-				
-				



Arrays

Find the Frequency of Each Element in an Array

```
import java.util.HashMap;
import java.util.Map;

public class FrequencyOfElements {
    public static void main(String[] args) {
        int[] numbers = {2, 2, 3, 4, 5, 5, 5, 3, 2, 4};
        Map<Integer, Integer> frequencyMap = new HashMap<>();

        for (int number : numbers) {
            frequencyMap.put(number, frequencyMap.getOrDefault(number,
0) + 1);
        }

        for (Map.Entry<Integer, Integer> entry :
frequencyMap.entrySet()) {
            System.out.println("Element: " + entry.getKey() + "
```

```

Frequency: " + entry.getValue());
    }
}
}

```

Duplicate Elements of an Array

```

import java.util.HashSet;

public class DuplicateElements {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 1, 2};
        HashSet<Integer> seen = new HashSet<>();
        HashSet<Integer> duplicates = new HashSet<>();

        for (int number : numbers) {
            if (!seen.add(number)) {
                duplicates.add(number);
            }
        }

        System.out.println("Duplicate elements: " + duplicates);
    }
}

```

Elements of an Array in Reverse Order

```

public class ReverseArray {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};

        System.out.print("Reversed Array: ");
        for (int i = numbers.length - 1; i >= 0; i--) {
            System.out.print(numbers[i] + " ");
        }
    }
}

```

Print the Largest Element in an Array

```

public class LargestElement {

```

```

public static void main(String[] args) {
    int[] numbers = {12, 34, 10, 1};
    int max = numbers[0];

    for (int number : numbers) {
        if (number > max) {
            max = number;
        }
    }

    System.out.println("Largest Element: " + max);
}
}

```

Print the Smallest Element in an Array

```

public class SmallestElement {
    public static void main(String[] args) {
        int[] numbers = {12, 34, 10, 1};
        int min = numbers[0];

        for (int number : numbers) {
            if (number < min) {
                min = number;
            }
        }

        System.out.println("Smallest Element: " + min);
    }
}

```

Sum of All Items of the Array

```

public class SumOfArray {
    public static void main(String[] args) {
        int[] numbers = {12, 34, 10};
        int sum = 0;

        for (int number : numbers) {
            sum += number;
        }
    }
}

```

```
        System.out.println("Sum of all elements: " + sum);
    }
}
```

Ascending Order Sort of the Array Elements.

```
import java.util.Arrays;

public class SortArray {
    public static void main(String[] args) {
        int[] numbers = {5, 3, 8, 1};

        Arrays.sort(numbers);

        System.out.print("Sorted Array: ");
        for (int number : numbers) {
            System.out.print(number + " ");
        }
    }
}
```

Find Second Largest Number in an Array

```
import java.util.Arrays;

public class SecondLargestNumber {
    public static void main(String[] args) {
        int[] numbers = {12, 34, 10, 1};

        Arrays.sort(numbers);

        System.out.println("Second Largest Number: " +
            numbers[numbers.length - 2]);
    }
}
```

Second Smallest Number in an Array

```
import java.util.Arrays;

public class SecondSmallestNumber {
```



```

public static void main(String[] args) {
    int[] numbers = {12, 34, 10, 1};

    Arrays.sort(numbers);

    System.out.println("Second Smallest Number: " + numbers[1]);
}
}

```

Odd and Even Numbers from an Array

```

public class OddEvenNumbers {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4};

        System.out.print("Even Numbers: ");
        for (int number : numbers) {
            if (number % 2 == 0) {
                System.out.print(number + " ");
            }
        }

        System.out.print("\nOdd Numbers: ");
        for (int number : numbers) {
            if (number % 2 != 0) {
                System.out.print(number + " ");
            }
        }
    }
}

```



Remove All White Spaces in a String

```

public class RemoveWhiteSpaces {

```

```

public static void main(String[] args) {
    String str = " Geeks for Geeks ";

    str = str.replaceAll("\\s", "");

    System.out.println("String without spaces: '" + str + "'");
}
}

```

Prove that String Objects are Immutable

```

public class StringImmutabilityTest {
    public static void main(String[] args) {
        String str = "Hello";

        // Attempt to change the string
        str.concat(", World!");

        // Original string remains unchanged
        System.out.println(str); // Output will be 'Hello'
    }
}

```

Count the Number of Words in a String

```

public class CountWordsInString {
    public static void main(String[] args) {
        String str = "Hello world! Welcome to Java.";

        String[] words = str.split("\\s+");

        System.out.println("Number of words: " + words.length);
    }
}

```

String is a Palindrome

```

public class PalindromeCheck {
    public static void main(String[] args) {
        String str = "madam";
    }
}

```

```

        String reversedStr = new
StringBuilder(str).reverse().toString();

        if (str.equals(reversedStr)) {
            System.out.println(str + " is a palindrome.");
        } else {
            System.out.println(str + " is not a palindrome.");
        }
    }
}

```

Reverse a String

```

public class ReverseString {
    public static void main(String[] args) {
        String str = "Hello";

        String reversedStr = new
StringBuilder(str).reverse().toString();

        System.out.println("Reversed String: " + reversedStr);
    }
}

```

Remove Leading Zeros

```

public class RemoveLeadingZeros {
    public static void main(String[] args) {
        String str = "00012345";

        str = str.replaceFirst("^0+(?!$)", "");

        System.out.println("String without leading zeros: '" + str +
        "'");
    }
}

```

First Letter of Each Word in a String

```

public class FirstLetterOfEachWord {
    public static void main(String[] args) {

```

```

        String str = "Hello World from Java";
        StringBuilder firstLetters = new StringBuilder();

        for (String word : str.split("\\s+")) {
            firstLetters.append(word.charAt(0));
        }

        System.out.println("First letters: " +
firstLetters.toString());
    }
}

```

Longest Substring Without Repeating Characters

```

import java.util.HashSet;

public class LongestSubstringLength {

    public static void main(String[] args) {

        String s = "abcabcbb";
        int maxLength = findLongestSubstring(s);

        System.out.println("Length of longest substring without repeating
characters: " + maxLength);
    }

    private static int findLongestSubstring(String s) {

        HashSet<Character> set = new HashSet<>();
        int leftPointer = 0;
        int maxLength = 0;

        for (int rightPointer = 0; rightPointer < s.length();
rightPointer++) {

            while (set.contains(s.charAt(rightPointer))) {

                set.remove(s.charAt(leftPointer));
                leftPointer++;
            }

            set.add(s.charAt(rightPointer));
            maxLength = Math.max(maxLength, rightPointer - leftPointer +

```

```

1);
    }

    return maxLength;
}
}

```

✓ Design Patterns

Design patterns are standard solutions to common problems in software design

Singleton Pattern

The Singleton pattern ensures that a class has only one instance and provides a global point of access to it.

Example:

```

public class Singleton {
    private static Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}

```

Factory Pattern

The Factory pattern defines an interface for creating an object but lets subclasses alter the type of objects that will be created.

Example:

```

public interface Shape {
    void draw();
}

public class Circle implements Shape {
    public void draw() {
        System.out.println("Drawing Circle");
    }
}

```

```

}

public class Rectangle implements Shape {
    public void draw() {
        System.out.println("Drawing Rectangle");
    }
}

public class ShapeFactory {
    public Shape getShape(String shapeType) {
        if (shapeType == null) {
            return null;
        }
        if (shapeType.equalsIgnoreCase("CIRCLE")) {
            return new Circle();
        } else if (shapeType.equalsIgnoreCase("RECTANGLE")) {
            return new Rectangle();
        }
        return null;
    }
}

```

Decorator Pattern

The Decorator pattern allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class.

Example:

```

public interface Car {
    void assemble();
}

public class BasicCar implements Car {
    public void assemble() {
        System.out.print("Basic Car.");
    }
}

public class CarDecorator implements Car {
    protected Car decoratedCar;

    public CarDecorator(Car c) {
        this.decoratedCar = c;
    }
}

```

```

public void assemble() {
    this.decoratedCar.assemble();
}
}

public class SportsCar extends CarDecorator {
    public SportsCar(Car c) {
        super(c);
    }

    public void assemble() {
        super.assemble();
        System.out.print(" Adding features of Sports Car.");
    }
}

```

✓ Count vowels and consonants in a String

aeiou

input = pramod

vol = 2

```

# Function to count vowels and consonants in a string
def count_vowels_and_consonants(input_string):
    # Convert the input string to lowercase to handle both uppercase
    and lowercase characters
    input_string = input_string.lower()

    # Initialize counters for vowels and consonants
    vowel_count = 0
    consonant_count = 0

    # Define a set of vowels
    vowels = "aeiou"

    # Iterate through each character in the input string
    for char in input_string:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

```

```
        return vowel_count, consonant_count

# Input string
input_string = "pramod"

# Call the function to count vowels and consonants
vowel_count, consonant_count =
count_vowels_and_consonants(input_string)

# Print the results
print("Vowel Count:", vowel_count)
print("Consonant Count:", consonant_count)
```

Leetcode 15 Easy Challenges

15 easy LeetCode problems along with their solutions in Java:

1. Two Sum

Problem Statement:

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

Java Solution:

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement)) {
            return new int[] { map.get(complement), i };
        }
        map.put(nums[i], i);
    }
    throw new IllegalArgumentException("No two sum solution");
}
```


2. Reverse Integer

Problem Statement:

Reverse digits of an integer.

Java Solution:

```
public int reverse(int x) {
    int result = 0;
    while (x != 0) {
        int pop = x % 10;
        x /= 10;
        if (result > Integer.MAX_VALUE / 10 || (result ==
Integer.MAX_VALUE / 10 && pop > 7)) {
            return 0;
        }
        if (result < Integer.MIN_VALUE / 10 || (result ==
Integer.MIN_VALUE / 10 && pop < -8)) {
            return 0;
        }
        result = result * 10 + pop;
    }
    return result;
}
```

3. Palindrome Number

Problem Statement:

Determine whether an integer is a palindrome.

Java Solution:

```
public boolean isPalindrome(int x) {
    if (x < 0) return false;
    int original = x;
    int reversed = 0;
    while (x != 0) {
        int digit = x % 10;
        reversed = reversed * 10 + digit;
        x /= 10;
    }
    return original == reversed;
}
```

4. Roman to Integer

Problem Statement:

Convert a Roman numeral to an integer.

Java Solution:

```
public int romanToInt(String s) {
    Map<Character, Integer> values = new HashMap<>();
    values.put('I', 1);
    values.put('V', 5);
    values.put('X', 10);
    values.put('L', 50);
    values.put('C', 100);
    values.put('D', 500);
    values.put('M', 1000);

    int result = 0;
    for (int i = 0; i < s.length(); i++) {
        int currentVal = values.get(s.charAt(i));
        int nextVal = (i < s.length() - 1) ? values.get(s.charAt(i + 1))
: 0;

        if (currentVal < nextVal) {
            result -= currentVal;
        } else {
            result += currentVal;
        }
    }
    return result;
}
```

5. Longest Common Prefix

Problem Statement:

Find the longest common prefix string amongst an array of strings.

Java Solution:

```
public String longestCommonPrefix(String[] strs) {
    if (strs == null || strs.length == 0) return "";
    String prefix = strs[0];
    for (int i = 1; i < strs.length; i++) {
        while (strs[i].indexOf(prefix) != 0) {

```

```

        prefix = prefix.substring(0, prefix.length() - 1);
        if (prefix.isEmpty()) return "";
    }
}
return prefix;
}

```

6. Valid Parentheses

Problem Statement:

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

Java Solution:

```

public boolean isValid(String s) {
    Stack<Character> stack = new Stack<>();
    for (char c : s.toCharArray()) {
        if (c == '(' || c == '{' || c == '[') {
            stack.push(c);
        } else if (c == ')' && !stack.isEmpty() && stack.peek() == '(')
        {
            stack.pop();
        } else if (c == '}' && !stack.isEmpty() && stack.peek() == '{')
        {
            stack.pop();
        } else if (c == ']' && !stack.isEmpty() && stack.peek() == '[')
        {
            stack.pop();
        } else {
            return false;
        }
    }
    return stack.isEmpty();
}

```

7. Merge Two Sorted Lists

Problem Statement:

Merge two sorted linked lists and return it as a new sorted list.

Java Solution:

```

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    if (l1 == null) return l2;
    if (l2 == null) return l1;

    if (l1.val < l2.val) {
        l1.next = mergeTwoLists(l1.next, l2);
        return l1;
    } else {
        l2.next = mergeTwoLists(l1, l2.next);
        return l2;
    }
}

```

8. Remove Duplicates from Sorted Array

Problem Statement:

Given a sorted array, remove the duplicates in-place such that each element appears only once and return the new length.

Java Solution:

```

public int removeDuplicates(int[] nums) {
    if (nums.length == 0) return 0;
    int uniqueCount = 1;
    for (int i = 1; i < nums.length; i++) {
        if (nums[i] != nums[i - 1]) {
            nums[uniqueCount] = nums[i];
            uniqueCount++;
        }
    }
    return uniqueCount;
}

```

9. Implement strStr()

Problem Statement:

Implement strStr() which locates a substring within a given string.

Java Solution:

```

public int strStr(String haystack, String needle) {
    if (needle.isEmpty()) return 0;
    for (int i = 0; i <= haystack.length() - needle.length(); i++) {

```

```

        if (haystack.substring(i, i + needle.length()).equals(needle)) {
            return i;
        }
    }
    return -1;
}

```

10. Count and Say

Problem Statement:

The count-and-say sequence is the sequence of integers with the first five terms as follows:
1, 11, 21, 1211, 111221, ...

Java Solution:

```

public String countAndSay(int n) {
    if (n <= 0) return "";
    String result = "1";
    for (int i = 1; i < n; i++) {
        StringBuilder sb = new StringBuilder();
        char prevChar = result.charAt(0);
        int count = 1;
        for (int j = 1; j < result.length(); j++) {
            char currentChar = result.charAt(j);
            if (currentChar == prevChar) {
                count++;
            } else {
                sb.append(count).append(prevChar);
                prevChar = currentChar;
                count = 1;
            }
        }
        sb.append(count).append(prevChar);
        result = sb.toString();
    }
    return result;
}

```

11. Maximum Subarray

Problem Statement:

Find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Java Solution:

```

public int maxSubArray(int[] nums) {
    int maxSum = nums[0];
    int currentSum = nums[0];
    for (int i = 1; i < nums.length; i++) {
        currentSum = Math.max(nums[i], currentSum + nums[i]);
        maxSum = Math.max(maxSum, currentSum);
    }
    return maxSum;
}

```

12. Climbing Stairs

Problem Statement:

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Java Solution:

```

public int climbStairs(int n) {
    if (n <= 2) return n;
    int prev1 = 1;
    int prev2 = 2;
    for (int i = 3; i <= n; i++) {
        int current = prev1 + prev2;
        prev1 = prev2;
        prev2 = current;
    }
    return prev2;
}

```

13. Plus One

Problem Statement:

Given a non-empty array of digits representing a non-negative integer, increment one to the integer.

Java Solution:

```

public int[] plusOne(int[] digits) {
    int n = digits.length;

```

```

    for (int i = n - 1; i >= 0; i--) {
        if (digits[i] < 9) {
            digits[i]++;
            return digits;
        }
        digits[i] = 0;
    }
    int[] newNumber = new int[n + 1];
    newNumber[0] = 1;
    return newNumber;
}

```

14. Add Binary

Problem Statement:

Given two binary strings, return their sum (also a binary string).

Java Solution:

```

public String addBinary(String a, String b) {
    StringBuilder result = new StringBuilder();
    int carry = 0;
    int i = a.length() - 1, j = b.length() - 1;
    while (i >= 0 || j >= 0 || carry > 0) {
        int sum = carry;
        if (i >= 0) {
            sum += a.charAt(i--) - '0';
        }
        if (j >= 0) {
            sum += b.charAt(j--) - '0';
        }
        result.insert(0, sum % 2);
        carry = sum / 2;
    }
    return result.toString();
}

```

15. Sqrt(x)

Problem Statement:

Compute and return the square root of x, where x is a non-negative integer.

Java Solution:

```
public int mySqrt(int x) {  
    if (x == 0) return 0;  
    int left = 1, right = x;  
    while (left <= right) {  
        int mid = left + (right - left) / 2;  
        if (mid <= x / mid && (mid + 1) > x / (mid + 1)) {  
            return mid;  
        } else if (mid > x / mid) {  
            right = mid - 1;  
        } else {  
            left = mid + 1;  
        }  
    }  
    return -1; // This should not happen for non-negative x.  
}
```

TheTestingAcc